GPy crash course

Alan Saul

University of Sheffield

Outline

- GPy project
- Key components
- Demos
- Parameters
- Models
- Kernels
- Inference methods
- Likelihoods

GPy project

- Open source Gaussian process framework written in Python.
- Built in a modular way designed to be easily extended and integrated.
- Primarily developed at the University of Sheffield under supervision of Neil Lawrence.
- Recently has been attracting lots of pull requests and bug reports as well as requests for additional models/help implementing new models, which is great, keep them coming!
- Written in Python with some bottlenecks sped up with Cython.
- https://github.com/SheffieldML/GPy

Todays objectives

- Gentle introduction to the project.
- More detail on the project layout for those interested on extending it or building upon it.
- Start implementing own projects, where I can help assist with any questions you might have.
- Gather collection of most wanted missing features.

Main Features

- Nice handling of model parameters
- GP regression
- Sparse GP regression
- Gaussian process latent variable model and Bayesian version, MRD etc.
- EP and Laplace along with a number of likelihoods
- Number of kernels along with easy ways of combining kernels

Installation

- We strongly recommend that you use anaconda python distribution: https://www.continuum.io/downloads
- This is essentially a package manager and environment for python modules used for scientific work.
- Quick and easy to install and uninstall highly optimized versions (including MKL support) of scientific libraries.
- Since we are dealing with packages that use underlying Fortran/C code, it is nice to have an environment that handles the issues with this for us.

Developer installation

For troubleshooting it is best to install from source

```
conda install pip
conda update scipy
git clone https://github.com/SheffieldML/GPy.git
cd GPy
git checkout devel
pip install –e . # This installs using pip in developer mode
nosetests GPy/testing
```

Demo

Demo time!

Extending GPy

GPy although not as feature rich as some other packages is built to be easily built upon and added to.

This next section will be dedicated showing you what you need to know to extend the existing framework (and submit pull requests!)

Parameters

The parameterization framework that has been refactored to `paramz` allows the magic to happen that makes creating new models, likelihoods, and kernels easy. Almost every object in GPy inherits from Parameterized.

Parameter values are held in a numpy array in `the_object.param_array`

When a new model/kernel/likelihood is created, we first create a Param object, and then link it to the object, which informs the object of its existence, and its parents etc.

The syntax of a new parameter is: `GPy.Param(self, name, input_array, default_constraint=None)`

```
from paramz.transformations import Logexp
self.var = GPy.Param('var', val, Logexp())
self.link_parameters(self.var)
```

Implementing a kernel

If the kernel is stationary, inherit from `Stationary` in `GPy.kern.src.stationary.py` If the kernel is non-stationary, inherit from `Kern` in `GPy.kern.src.kern.py`

Key functions that need to be implemented

- `K(X,X2)` - given two vectors, calculate the full covariance matrix - if X2 is None we assume that X2 = X
- `Kdiag(X)` - diagonal of above
- `update_gradient_full(dL_dK, X, X2)` - Given matrix $\frac{dL}{dK(X,X2)}$ compute $\frac{dL}{d\theta_K}$

For latent variable models `gradients_X, gradients_X_diag` also need to be computed.

Models

- A model is essentially a container for kernels, inference methods, and likelihoods.
- Typically a model will have some parameters, or objects that contain parameters (e.g kernel), that need to be optimized.
- The purpose of the model is to store these parameters, run the chosen inference method on each optimization step, and set the new gradient for each parameter.
- They are informed whenever any of their child's parameters change, by `parameters_changed()` being called.

Implementing models

- Parameters are made, and linked in the `__init__` method. For example, lets say our model had some parameters called `alpha`, our init might look like:

```
class NewModel(GPy.core.GP):
    def __init__(self, X, Y, kernel, likelihood, inference_method)
        self.alpha = Param('alpha', np.zeros((5,1)))
        super(NewModel, self).__init__(X, Y, kernel, likelihood,
   inference_method, name='NewModel')
        self.link_parameter(self.alpha)
```

Implementing models

- `parameters_changed()` is called automatically upon any change to the model that might change the marginal likelihood (if the parameters changed).
- `parameters_changed()` usually makes a call to the `inference_method` which will return the log marginal likelihood, a posterior object, and a dict holding the various derivatives (`grad_dict['dL_dK']`).
- Gradients are set for the parameters, `upgrade_gradients_full` when a chain rule needs to be applied to set the gradient, `self.name_of_parameter.gradient = dL_dparameter` for setting parameters gradients directly.

Inference methods

Inference methods are simply an object with a method `inference` that:

- Returns the marginal likelihood, L
- Returns a `Posterior` object that caches relevant matrices generated whilst computing the marginal likelihood, that allow the posterior to be computed efficiently (Cholesky of $K + \sigma^2 I$)
- Returns the gradients of the marginal likelihood wrt relevant parameters (for kernel $\frac{dL}{dK}$, for likelihood parameters $\frac{dL}{d\theta_l}$)

Likelihoods

Currently available likelihoods include:

- Bernoulli
- Exponential
- Gaussian
- Poisson

- More are soon to come as they implemented but not merged.
- Currently not all work with EP, we would love pull requests for this!
- However likelihoods are easy to implement, particularly for the Laplace approximation.

Implementing likelihoods for Laplace

Required functions:

- logpdf_link ($\log p(y|\lambda(f))$)
- dlogpdf_dlink ($\frac{d \log p(y|\lambda(f))}{d\lambda(f)}$)
- d2logpdf_dlink2 ($\frac{d^2 \log p(y|\lambda(f))}{d\lambda(f)^2}$)
- d3logpdf_dlink3 ($\frac{d^3 \log p(y|\lambda(f))}{d\lambda(f)^3}$)
- samples ($y* \sim p(y^*|\lambda(f^*))$)
- dlogpdf_link_dtheta ($\frac{\log p(y|\lambda(f))}{d\theta}$)
- dlogpdf_dlink_dtheta ($\frac{d}{d\theta}\frac{d \log p(y|\lambda(f))}{d\lambda(f)}$)
- d2logpdf_dlink2_dtheta $\frac{d}{d\theta}\frac{d^2 \log p(y|\lambda(f))}{d\lambda(f)^2}$

Implementing likelihoods for EP

Requires:

- `moment_match_ep` (Moments match of the marginal approximation)
- samples ($y* \sim p(y^*|\lambda(f^*))$)

Thanks

Thanks for listening, now lets take a quick look at GPyOpt and then get down to personal projects.